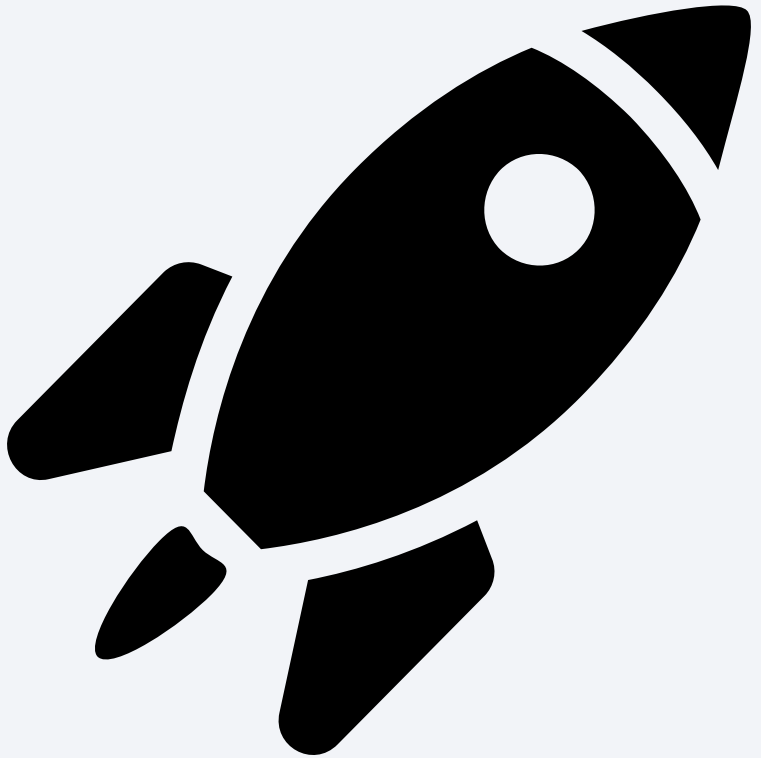
IBM Developer
SKILLS NETWORK

the Winning Space Race with Data Science

Annette M. Hynes
2024-Sep-26

Outline/Table of Contents

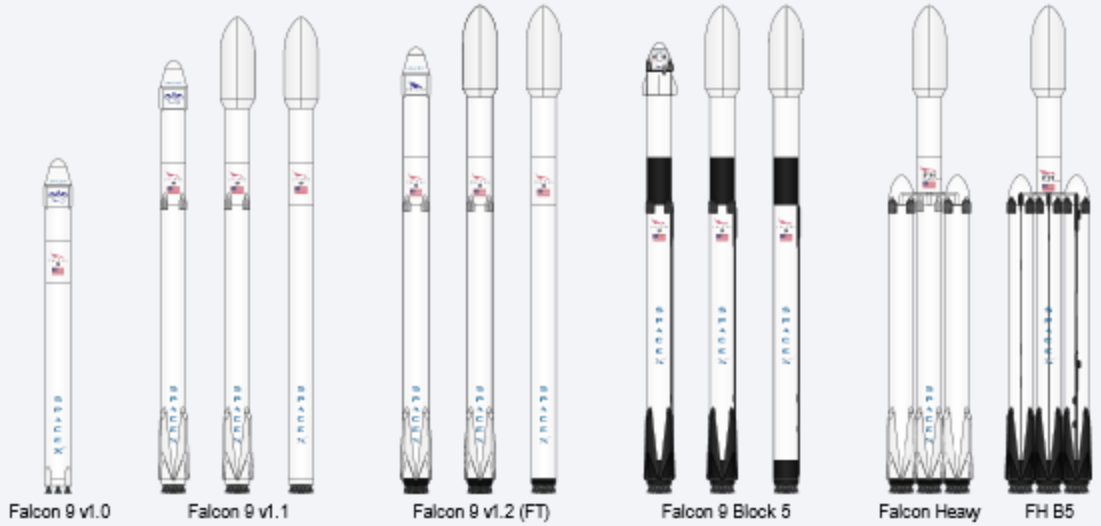


Executive Summary

- Objective
  - Predict the successful landing of the SpaceX Falcon 9 rocket. Being able to re-use the first stage of the rocket saves the company up to $1M per launch.
- Methodologies
  - Use the BeautifulSoup webscraping tool in Python and SQL to gather SpaceX launch data, convert the data to a pandas dataframe, and wrangle and explore the data to summarize launch outcomes in variable 'Class' using pandas and Matplotlib.
  - Use Folium and Plotly Dash to visualize and explore the data interactively.
  - Use machine learning to predict launch success by splitting the data into train and test sets and optimizing the hyperparameters for SVM, Decision Trees, Logistic Regression, and KNN.
- Results
  - Launch success increased with time. Payloads < 6000 kg were generally more successful.
  - ML models predict launch results with 83% accuracy. There is not enough data for the models to diverge from each other.

Introduction

- The SpaceX Falcon 9 rocket launches are advertised as $62 M.
- Rocket launches from other providers cost $165 M.
- Much of SpaceX's savings comes from re-using the first stage.
- Our goal is to predict if the Falcon 9 first stage will land successfully, thus determining the cost of a launch.



Section 1

Methodology

Methodology

- Data collection methodology:
  - Requested data from the SpaceX API.
  - Used BeautifulSoup to extract Falcon 9 data from Wikipedia.
- Perform data wrangling
  - Converted JSON data to a pandas array and got the booster version, launch site, etc.
  - Filtered data to include only Falcon 9 launches and replaced missing values with mean.
  - Parsed the scraped HTML tables into a dataframe.
  - Created a landing outcome label to help calculate success rate.

Methodology

- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Created a column for "Class" using numpy, standardized the data, and split the data into testing and training sets.
  - Found the best hyperparameters using grid search for logistic regression, support vector machine, and decision tree.

Data Collection

- Data sets were collected from:
  - SpaceX API
  - Webscraping Wikipedia

Data Collection – SpaceX API

1. Request rocket launch data from SpaceX API.

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

2. Convert JSON data to a pandas array.

```
data = pd.json_normalize(response.json())
```

3. Clean data with custom functions.

```
getBoosterVersion(data)
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

4. Create pandas dataframe using dictionary built from data.

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

```
df_launch = pd.DataFrame.from_dict(launch_dict)
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/jupyter-labs-spacex-data-collection-api.ipynb

Data Collection – SpaceX API

5. Filter data frame for Falcon 9 data.

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

6. Replace NaNs with mean values

```
# Calculate the mean value of PayloadMass column
mean_PM = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, mean_PM, inplace=True)
```

7. Export data table as CSV.

```
df_launch = pd.DataFrame.from_dict(launch_dict)
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/jupyter-labs-spacex-data-collection-api.ipynb

Data Collection - Scraping

1. Request data from Wikipedia static URL.

```
data = requests.get(static_url).text
```

2. Create a BeautifulSoup object from data request.

```
soup = BeautifulSoup(data,"html.parser")
```

3. Find all html tables in the BeautifulSoup object.

```
html_tables = soup.find_all('table')
```

4. Extract all the columns from the table.

```
th_list = first_launch_table.find_all('th')

for head in th_list:
    col = extract_column_from_header(head)
    if col is not None and len(col) > 0:
        column_names.append(col)
```

6. Initialize a dictionary with relevant launch data

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/jupyter-labs-webscraping.ipynb

Data Collection - Scraping

7. Parse the data to complete the dictionary

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
    # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)
            launch_dict['Version Booster'].append(bv)
```

```
# Launch Site
# TODO: Append the bv into launch_dict with key `Launch Site`
launch_site = row[2].a.string
print(launch_site)
launch_dict['Launch site'].append(launch_site)

# Payload
# TODO: Append the payload into launch_dict with key `Payload`
payload = row[3].a.string
print(payload)
launch_dict['Payload'].append(payload)

# Payload Mass
# TODO: Append the payload_mass into launch_dict with key `Payload mass`
payload_mass = get_mass(row[4])
print(payload_mass)
launch_dict['Payload mass'].append(payload_mass)

# Orbit
# TODO: Append the orbit into launch_dict with key `Orbit`
orbit = row[5].a.string
print(orbit)
launch_dict['Orbit'].append(orbit)

# Customer
# TODO: Append the customer into launch_dict with key `Customer`
customer = row[6].text.strip()
print(customer)
launch_dict['Customer'].append(customer)

# Launch outcome
# TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
launch_outcome = list(row[7].strings)[0]
print(launch_outcome)
launch_dict['Launch outcome'].append(launch_outcome)

# Booster landing
# TODO: Append the launch_outcome into launch_dict with key `Booster landing`
booster_landing = landing_status(row[8])
print(booster_landing)
launch_dict['Booster landing'].append(booster_landing)
```

9. Export data as a CSV file.

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

8. Convert the dictionary to a pandas data frame.

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/jupyter-labs-webscraping.ipynb

Data Wrangling

1. Load the data.

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/dat
df.head(10)
```

2. Check percentage of nulls for each column.

```
df.isnull().sum()/len(df)*100
```

3. Check data types of each column.

```
df.dtypes
```

4. Calculate the number of launches at each site.

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

5. Calculate the number of occurrences of each orbit

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

6. Calculate the number of occurrences for each outcome.

```
landing_outcomes = df['Outcome'].value_counts()
print(landing_outcomes)
```

```
Outcome
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: count, dtype: int64
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/labs-jupyter-spacex-Data%20wrangling.ipynb

Data Wrangling

7. Assign the bad outcomes to a list.

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

8. Assign an outcome label to landing_class:
0 = bad, 1 = successful

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for oc in df['Outcome']:
    if oc in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

9. Find the mean success rate.

```
df["Class"].mean()
```

```
np.float64(0.6666666666666666)
```

10. Export the landing data to a CSV file.

```
df.to_csv("dataset_part_2.csv", index=False)
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

- Graph summary:
  - Payload vs Flight Number scatter plot with class overlay—How do payload and flight number (a proxy of time) affect launch success?
  - Launch site versus Flight Number scatter plot with class overlay—How do launch site and flight number affect launch success?
  - Launch site versus Payload scatter plot with class overlay—How do launch site and flight number affect launch success?
  - Mean success for each Orbit bar chart—How successful is each type of orbit?
  - Orbit versus Flight Number scatter plot with class overlay—How do orbit and flight number affect launch success?

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/edadataviz.ipynb

EDA with SQL

- Query summary:
  - List unique launch sites.
  - Display 5 records of launch sites beginning with the string 'CCA'.
  - Calculate total payload mass carried by boosters launched by NASA (CRS).
  - Calculate the average payload mass carried by booster version F9 v1.1.
  - List the date of the first succesful landing on a ground pad.
  - List the boosters which have success in drone ship and have 4000 < payload mass < 6000.
  - Calculate the total number of successes and failures.
  - Using a subquery, list the names of the boosters which have carried the maximum payload mass.
  - Display the month names, failed drone ship landings, booster versions, and launch_site for 2015.
  - Rank the count of landing outcomes for 2010-06-04--2017-03-20, in descending order.

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- Site map object summary:
  - Labeled markers of launch sites to identify launch sites.
  - Marker cluster to expand/contract markers for visibility.
  - Colored markers to identify successful and failed launches.
  - Mouse position to get latitude and longitude using the mouse.
  - PolyLine and distance markers to draw and calculate distances from a launch site of interest to points of interest.

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- Plot summary
  - Pie chart
    - Shows the percentage of successful launches among launch sites when all sites are chosen.
    - Shows the percentage of successful versus failed launches for a particular site when chosen.
  - Scatter plot
    - Launch success class versus payload mass.
    - Sliders allow user to select range of payload mass.
    - Color denotes booster version.

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/spacex_dash_app.py

Predictive Analysis (Classification)

Model summary: Used machine learning to predict launch success by splitting the data into train and test sets and optimizing the hyperparameters for SVM, Classification Trees, Logistic Regression, and k Nearest Neighbors.

1. Import the data and then create a NumPy array from "Class."

```
Y = data['Class'].to_numpy()
type(Y)
```

2. Standardize the data.

```
transform = preprocessing.StandardScaler()
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
```

3. Split the data into training and testing sets.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 2)
```

4. Fit hyperparameters for a logistic regression using grid search and calculate accuracy..

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
gscv = GridSearchCV(lr, parameters, scoring = 'accuracy',cv = 10)
logreg_cv = gscv.fit(X_train, Y_train)
```

```
logreg_score = logreg_cv.score(X_test, Y_test)
print(logreg_score)
```

```
0.8333333333333334
```

5. Fit hyperparameters for a support vector machine using grid search and calculate accuracy.

```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
grid = GridSearchCV(svm, parameters, cv=10)
svm_cv = grid.fit(X_train, Y_train)
```

```
print("tuned hyperparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
svm_score = svm_cv.score(X_test, Y_test)
print(svm_score)
```

```
0.8333333333333334
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Predictive Analysis (Classification)

6. Fit hyperparameters for a decision tree using grid search and calculate accuracy.

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
tree_score = tree_cv.score(X_test, Y_test)
print(tree_score)
```

```
0.8333333333333334
```

7. Fit hyperparameters for k nearest neighbors using grid search and calculate accuracy.

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

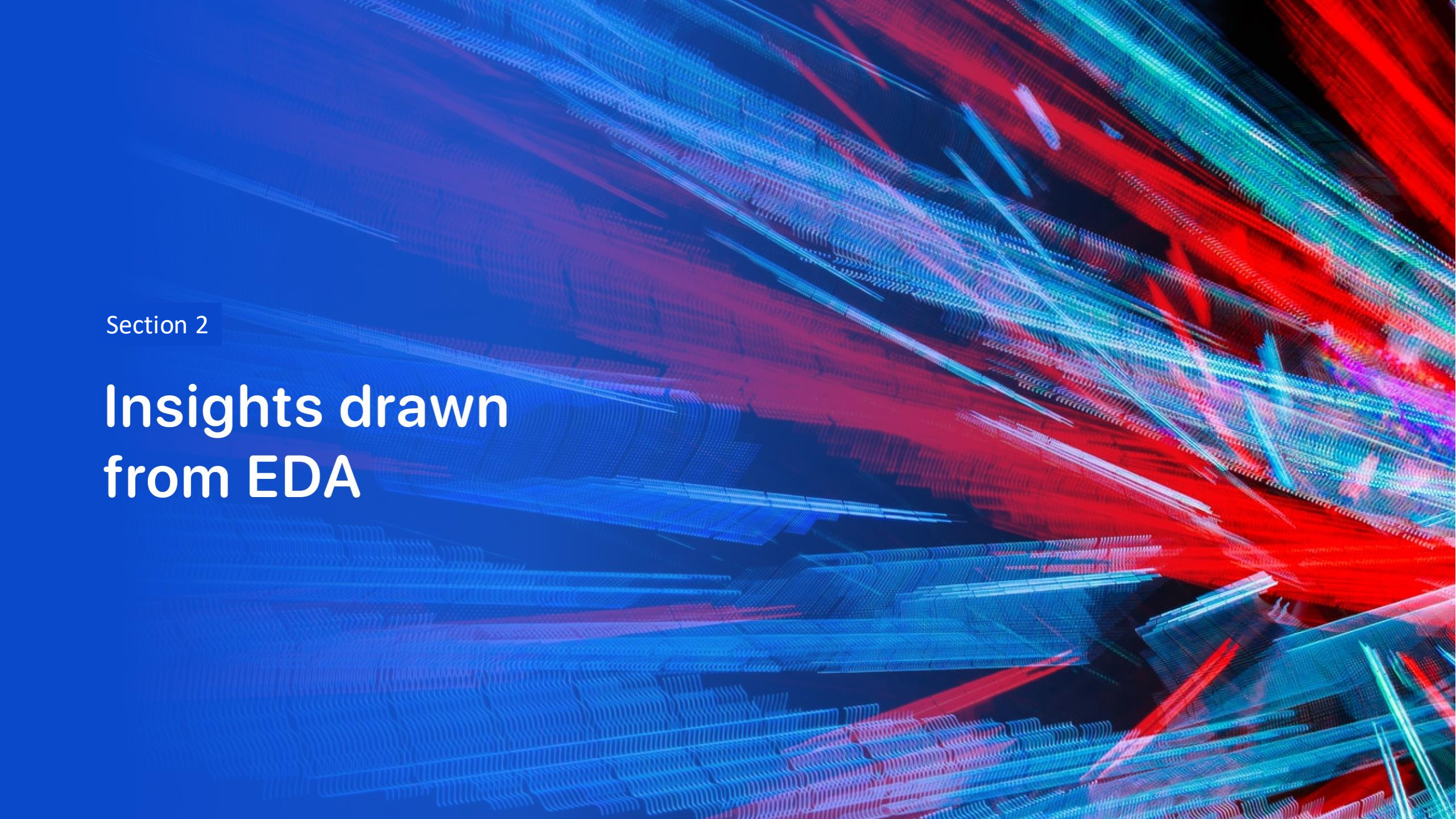
```
knn_score = knn_cv.score(X_test, Y_test)
print(knn_score)
```

```
0.8333333333333334
```

https://github.com/ANetTow/testrepo/blob/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

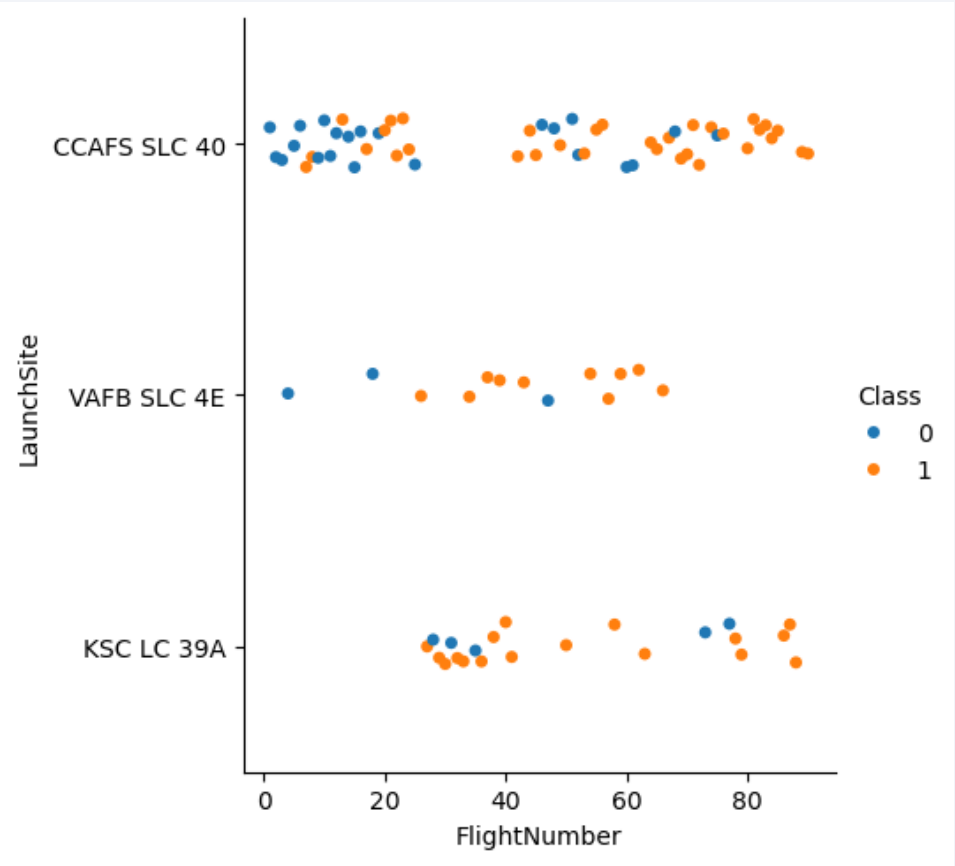
Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

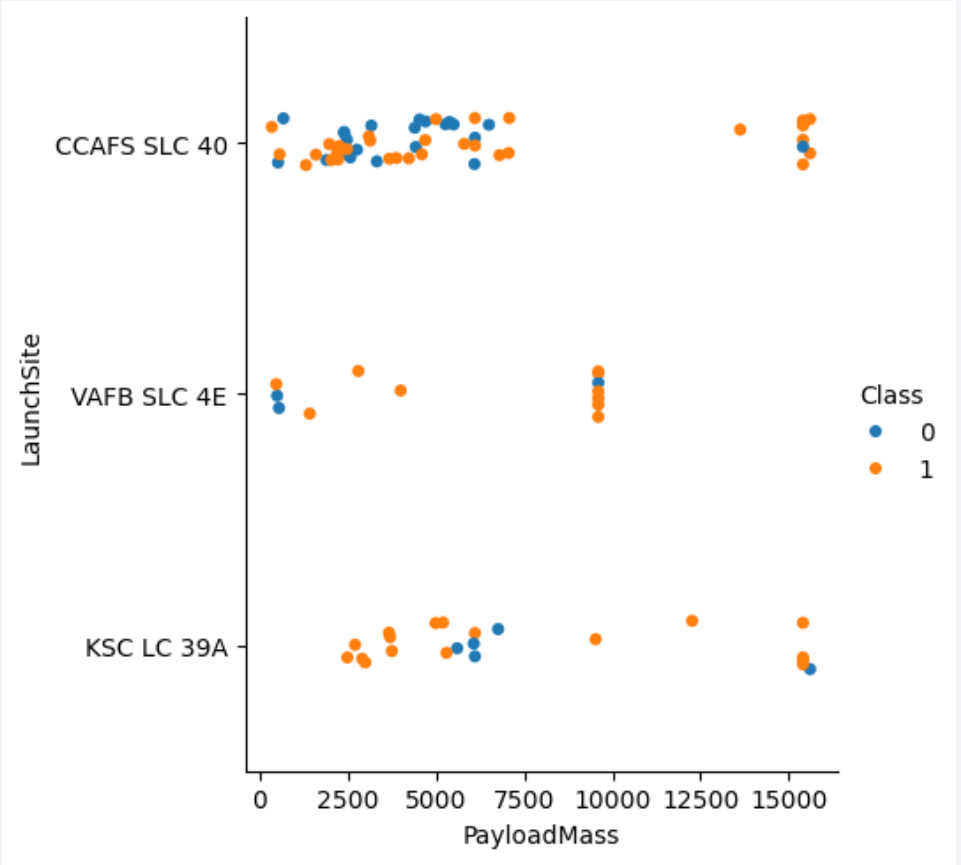
Flight Number vs. Launch Site

- Scatter plot of Launch Site (Y) vs. Flight Number (X) colored by launch success class.
- Blue = 0 (failure)
- Orange = 1 (success)
- A large number of the failures were early (flight number < 20) and at CCAFS SLC 40.



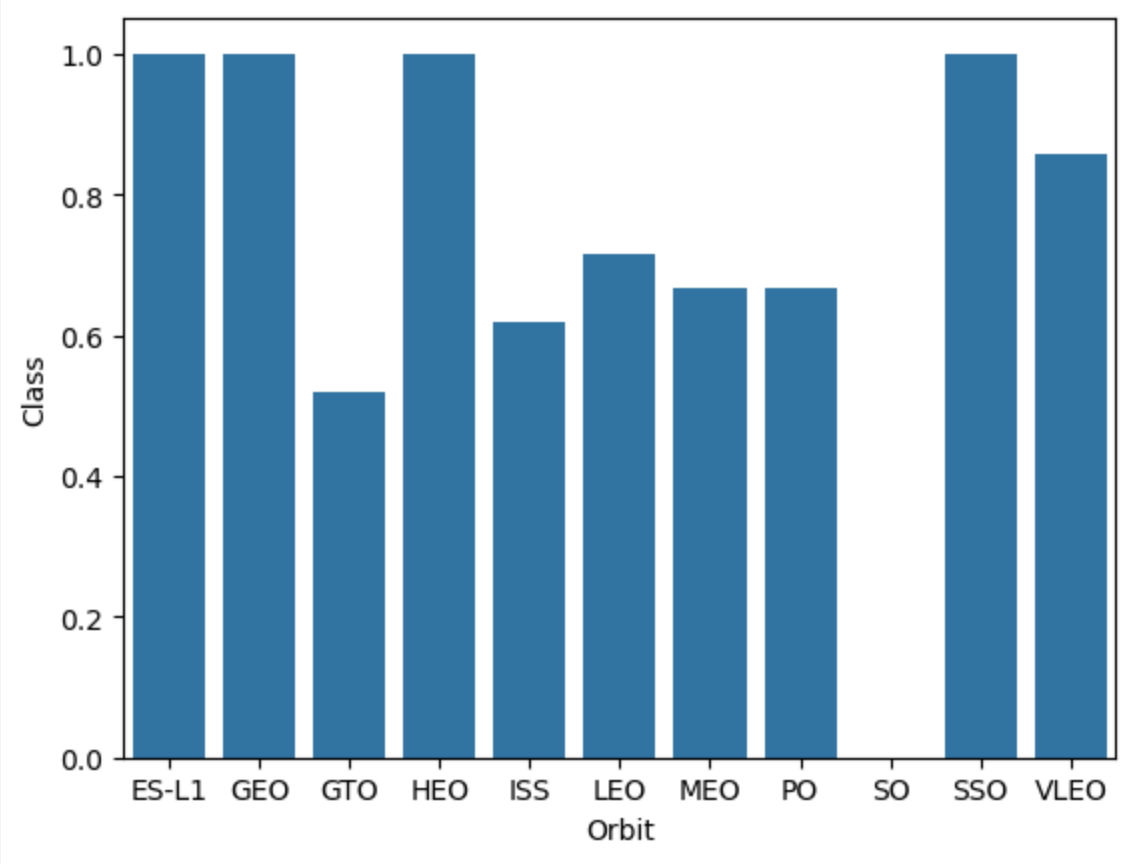
Payload vs. Launch Site

- Scatter plot of Launch Site (Y) vs. Payload Mass (kg, X) colored by launch success class (blue = failure, orange = success).
- Heavy payloads (> 10000) appear to be more successful.
- A histogram would be more effective than a scatter plot.

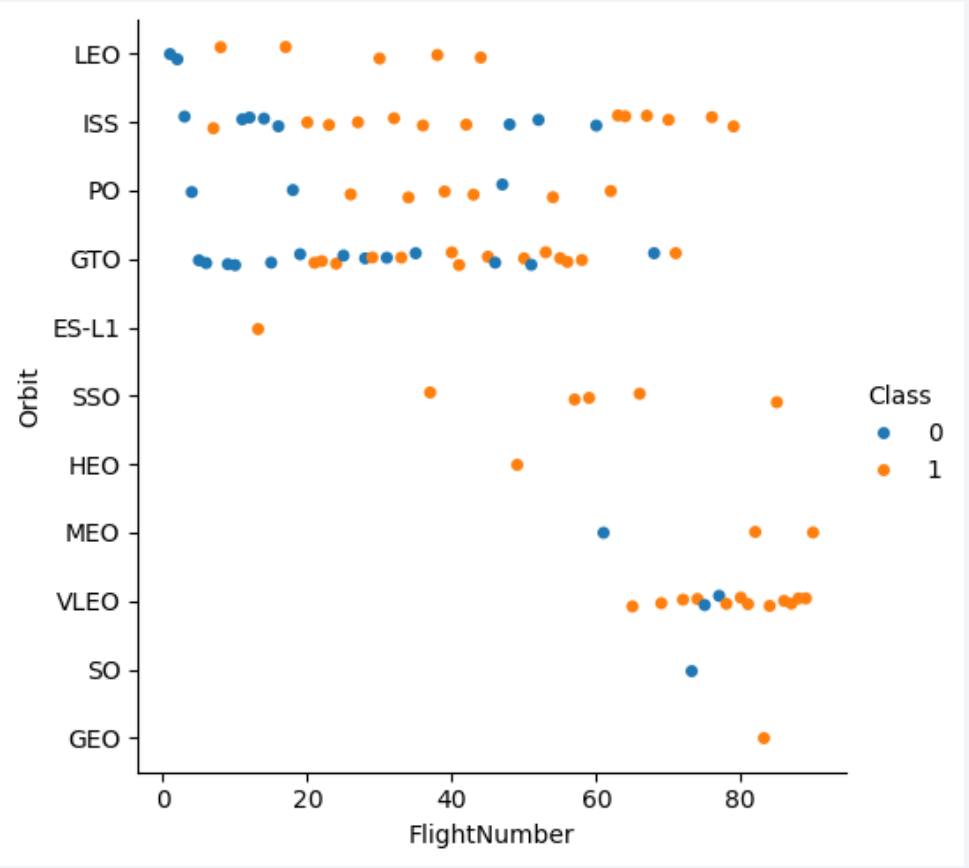


Success Rate vs. Orbit Type

- Bar chart for the success rate of each orbit type
- ES-L1 (1), GEO (1), HEO (1), and SSO (5) had perfect though limited success records.
- GTO was about 50% successful (27 launches) and SO completely failed (1 launch).



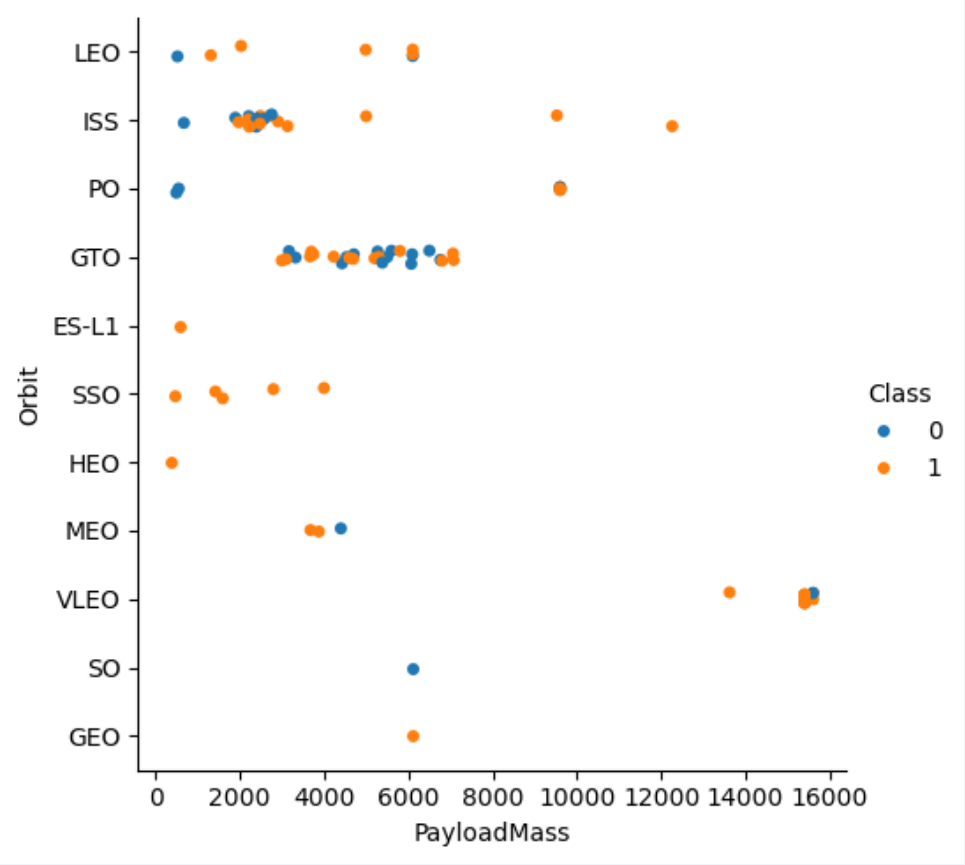
Flight Number vs. Orbit Type



- Scatter plot of Orbit type (Y) vs. Flight number (X) colored by flight success class.
- Most failures were early flights and GTO orbit, the most common orbit.
- Many Orbits have singular launches so we can't make conclusions based on them.

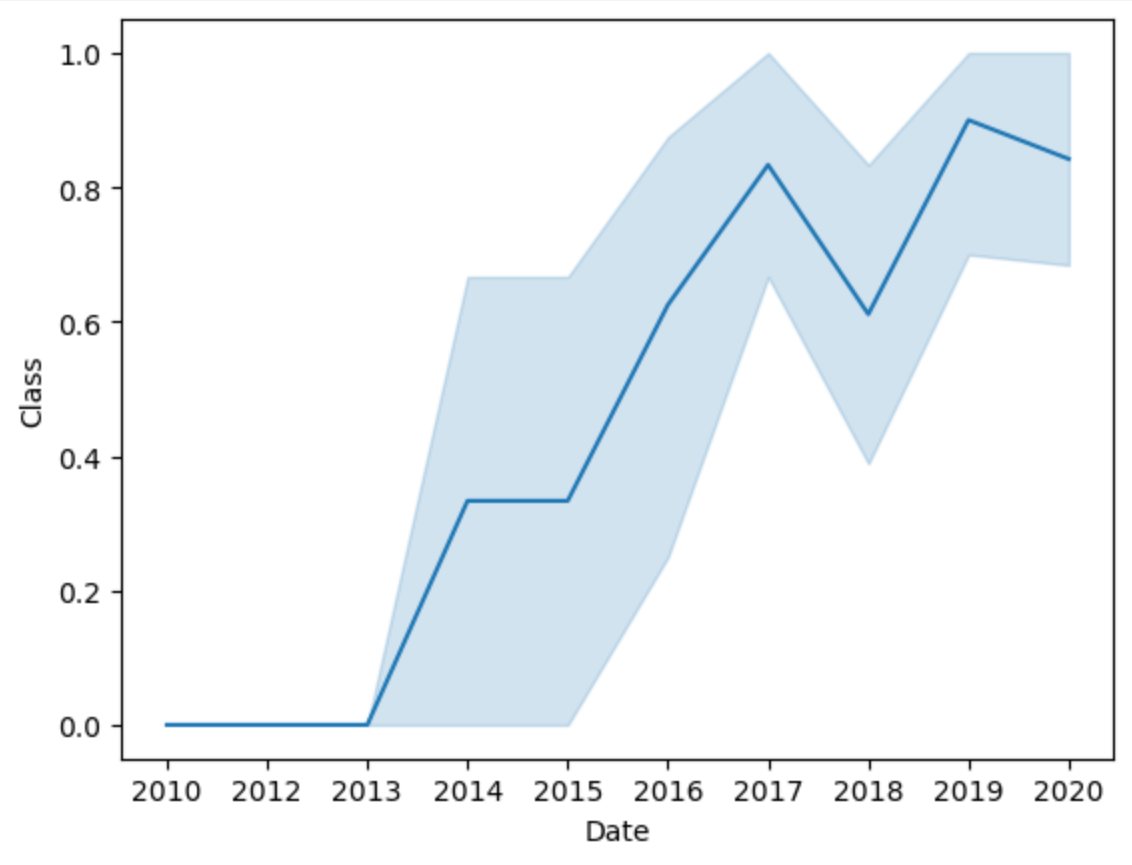
Payload vs. Orbit Type

- Scatter plot of Orbit (Y) vs. Payload Mass (kg, X) colored by success class.
- Each orbit has a specific payload range.
- Small payloads have more failures, but they have more launches in general.



Launch Success Yearly Trend

- Line plot of Mean Class (Y) versus Date (year, X).
- 2014 showed the first successful launches.
- Percentage of successes increased with time with a notable drop of success in 2018.



All Launch Site Names

- Find the names of the unique launch sites
- There were 4 launch sites.

```
%sql select distinct Launch_Site from SPACEXTABLE;
```

```
 * sqlite:///my_data1.db
Done.
```

Launch_Site
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`.
- The results are filtered using "like" and a wildcard character in 'CCA%'.

```
%sql select * from SPACEXTABLE where Launch_Site like 'CCA%' limit 5;
```

Python

```
 * sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- The sum of the payload carried by NASA was 45596 kg.

```
%sql select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where Customer like "NASA (CRS)"
```

```
 * sqlite:///my_data1.db
Done.
```

sum(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- The average payload mass carried by F9 v1.1 was 2534.67 kg.

```
%sql select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where Booster_Version like "F9 v1.1%"
```

```
 * sqlite:///my_data1.db
Done.
```

avg(PAYLOAD_MASS__KG_)
2534.6666666666665

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- The first successful landing on a ground pad was on 22-Dec-2015.

```
%sql select min(Date) from SPACEXTABLE where Landing_Outcome like 'Success (ground pad)';
```

```
 * sqlite:///my_data1.db
Done.
```

min(Date)
2015-12-22

Successful Drone Ship Landing with Payload between 400 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000.
- Use "and" for compound SQL queries. I found 4 boosters.

```
%sql select distinct Booster_Version from SPACEXTABLE
where Landing_Outcome like 'Success (drone ship)'
and PAYLOAD_MASS__KG_ > 4000 and PAYLOAD_MASS__KG_ < 6000
```

```
 * sqlite:///my_data1.db
Done.
```

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- There were 100 successful mission outcomes and 1 failure. Some instances when the stage is not recovered are still considered mission successes.

```
%sql select count(Mission_Outcome) from SPACEXTABLE where Mission_Outcome like "Success%";
```

```
 * sqlite:///my_data1.db
Done.
```

count(Mission_Outcome)
100

```
%sql select count(Mission_Outcome) from SPACEXTABLE \
where Mission_Outcome like "Failure%";
```

```
 * sqlite:///my_data1.db
Done.
```

count(Mission_Outcome)
1

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- 12 boosters carried the maximum payload mass.

```
%sql select distinct Booster_Version from SPACEXTABLE \
where (SELECT max(PAYLOAD_MASS__KG_) FROM SPACEXTABLE) = PAYLOAD_MASS__KG_;
```

```
 * sqlite:///my_data1.db
Done.
```

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- There were 2 failed landing outcomes for drone ships in 2015.

```
%sql select substr(Date, 6, 2) as Month, Landing_Outcome, Booster_Version, Launch_site from SPACEXTABLE \
where substr(Date, 0, 5) == '2015' \
and Landing_Outcome like "Failure (drone ship)";
```

```
 * sqlite:///my_data1.db
Done.
```

Month	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

- Present your query result with a short explanation here

```
%sql SELECT Landing_Outcome, count(Landing_Outcome) from SPACEXTABLE \
where substr(Date, 1, 4) || substr(Date, 6, 2) || substr(Date, 9, 2) between '20100604' and '20170320'\
group by Landing_Outcome order by count(Landing_Outcome) desc
```

```
 * sqlite:///my_data1.db
Done.
```

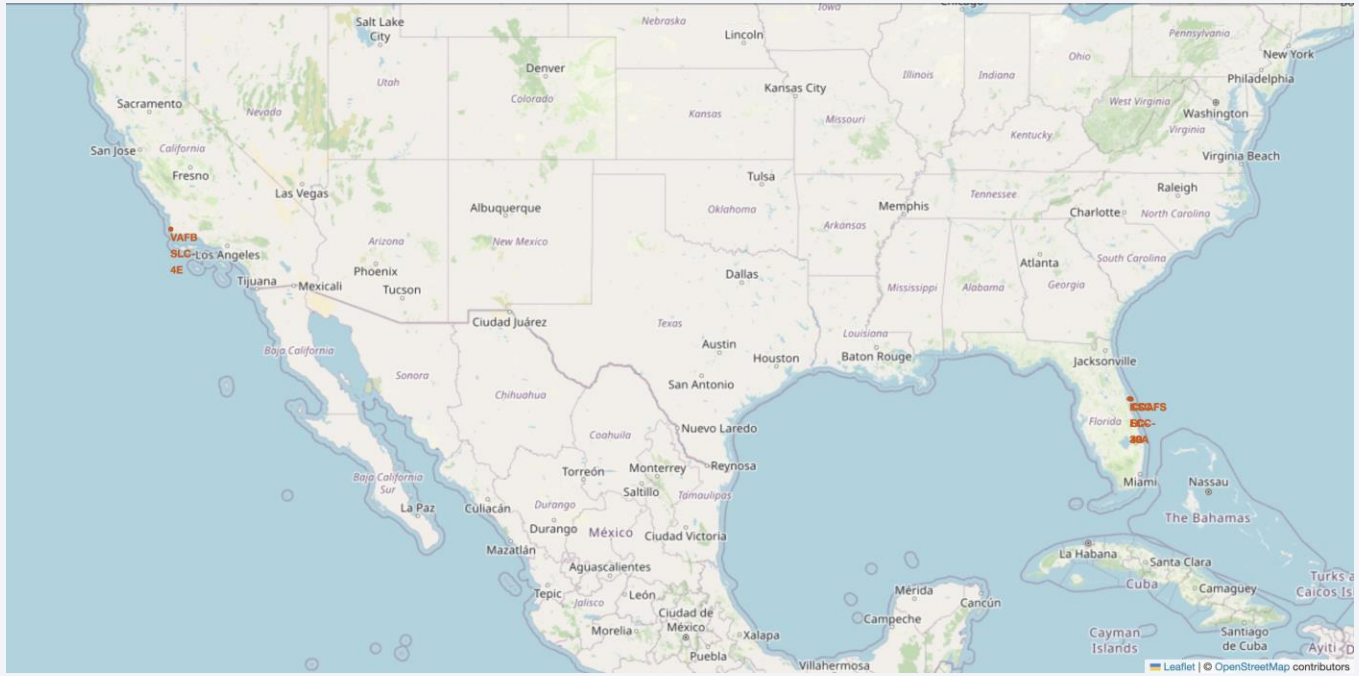
Landing_Outcome	count(Landing_Outcome)
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

Section 3

Launch Sites Proximities Analysis

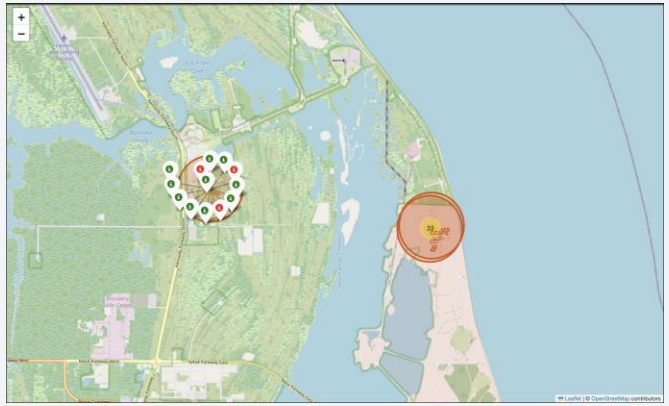
Launch Site Locations

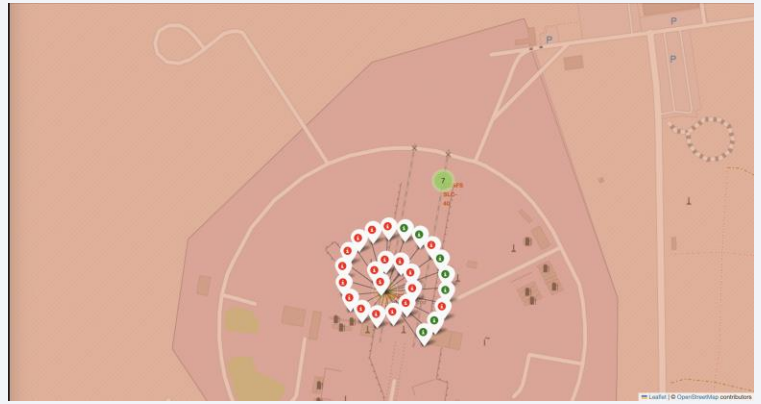
- Launch site locations were in California and Florida.
- Sites are coastal.
- Sites are as far south as possible in the continental United States.



Successful and Failed landings

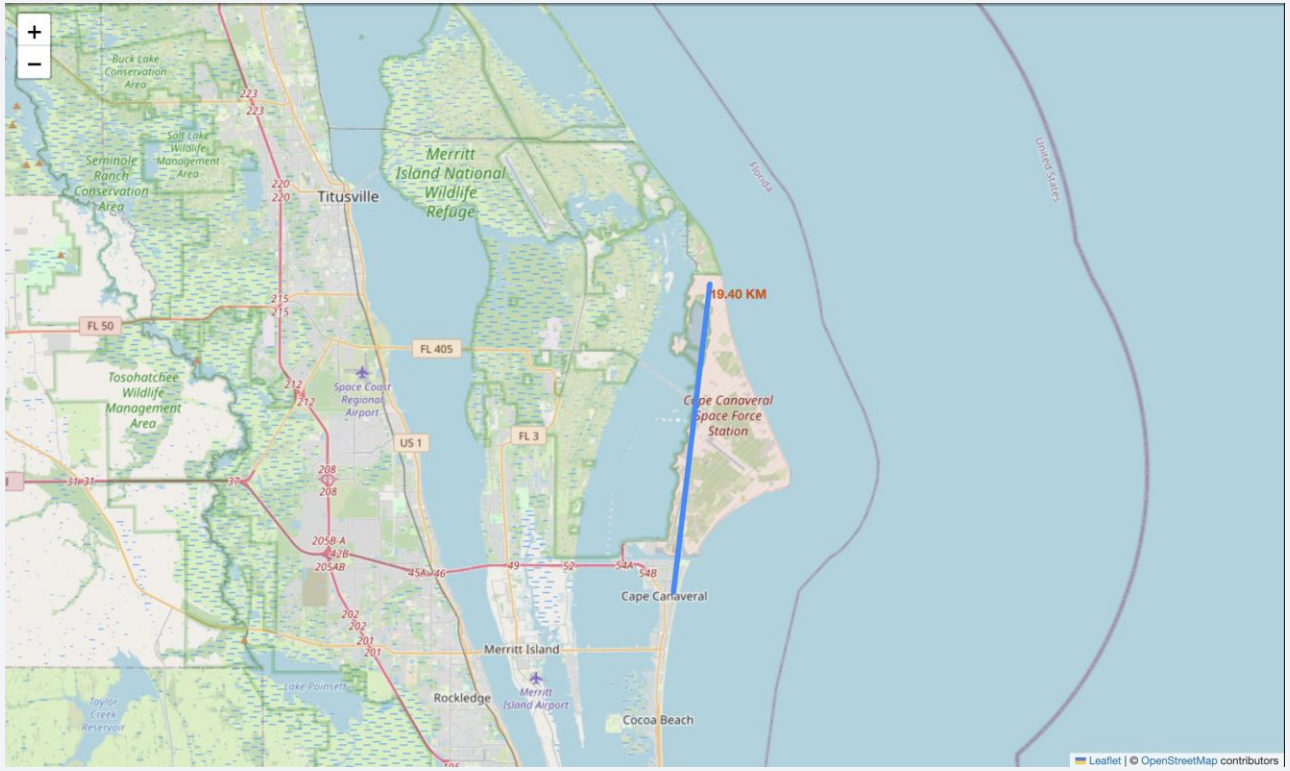
- Successful landings are marked in green, unsuccessful in red.
- Marker cluster allows you to expand/contract markers for visibility.
- More rockets were launched from Florida than from California.

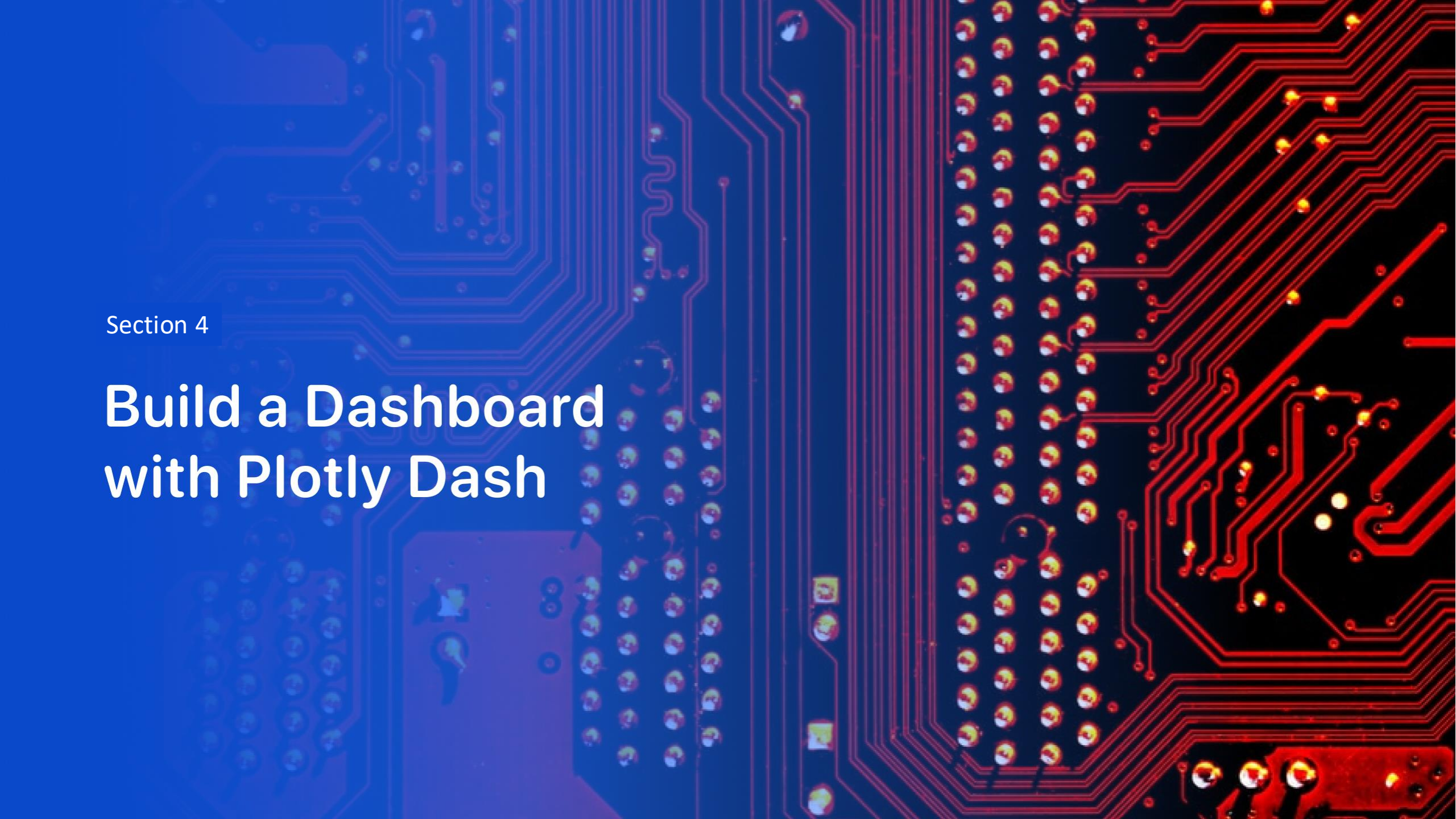




Measuring Distance

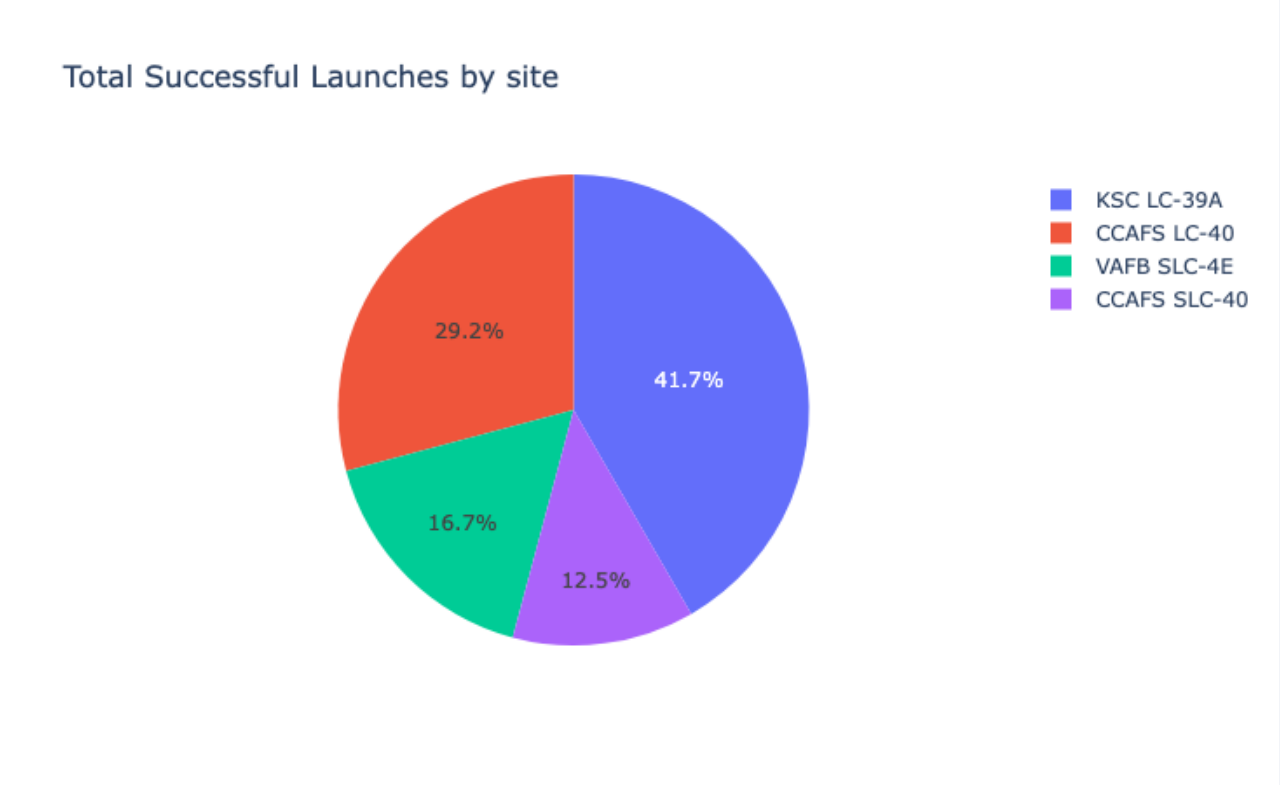
- The launch site is 19.4 km from Cape Canaveral.
- In general, launch sites are far from major highways and populated areas.




Section 4
Build a Dashboard with Plotly Dash

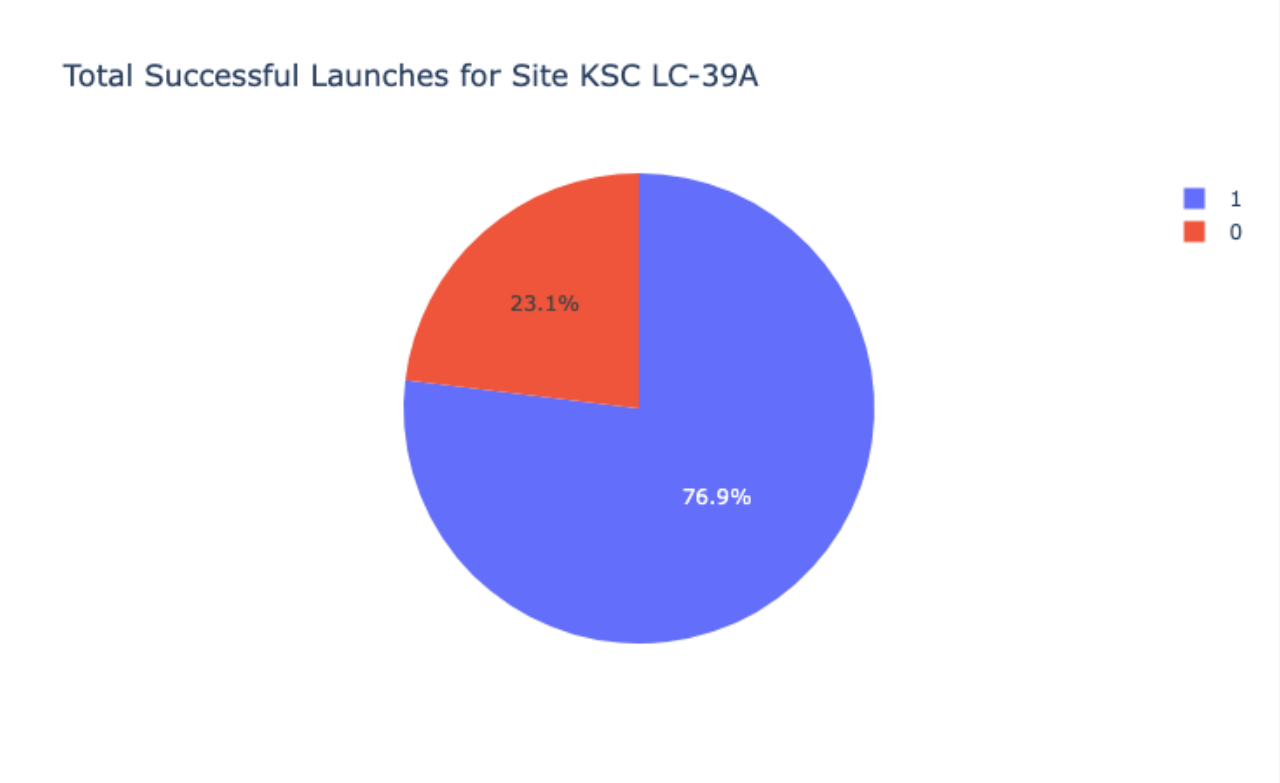
Total Successful Launches by Site

- KSC LC-39A had the most successful launches.
- From this graph we don't know the total launches, so it remains to be determined which site has the highest success rate.
- Pie charts are aesthetically appealing but not successful in communicating the differences between categories. A stacked bar chart would be more effective.



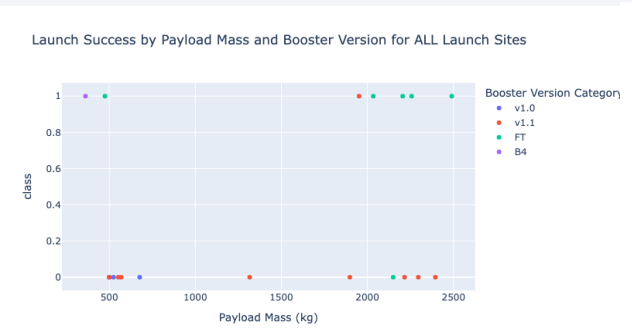
Percent success for KSC LC-39A

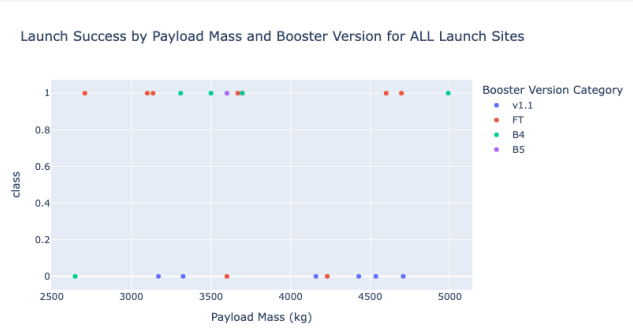
- KSC LC-39A had the highest success rate: 76.9%.

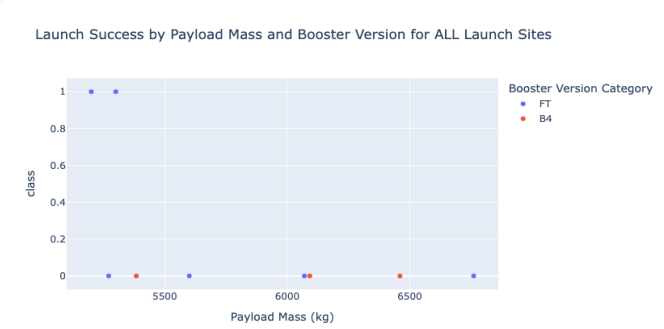


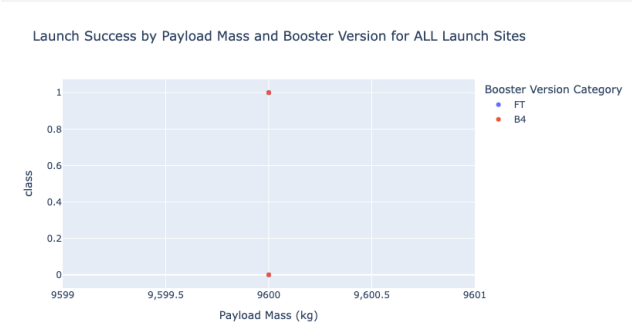
Launch success by payload: all sites

- Payload has been selected by divisions of 2500 kg.
- Payloads between about 2000—5000 kg appear to have the most success.
- FT appears to be the most successful booster category.
- Scatter plots are not effective in communicating what we need to learn. A histogram would be much better.





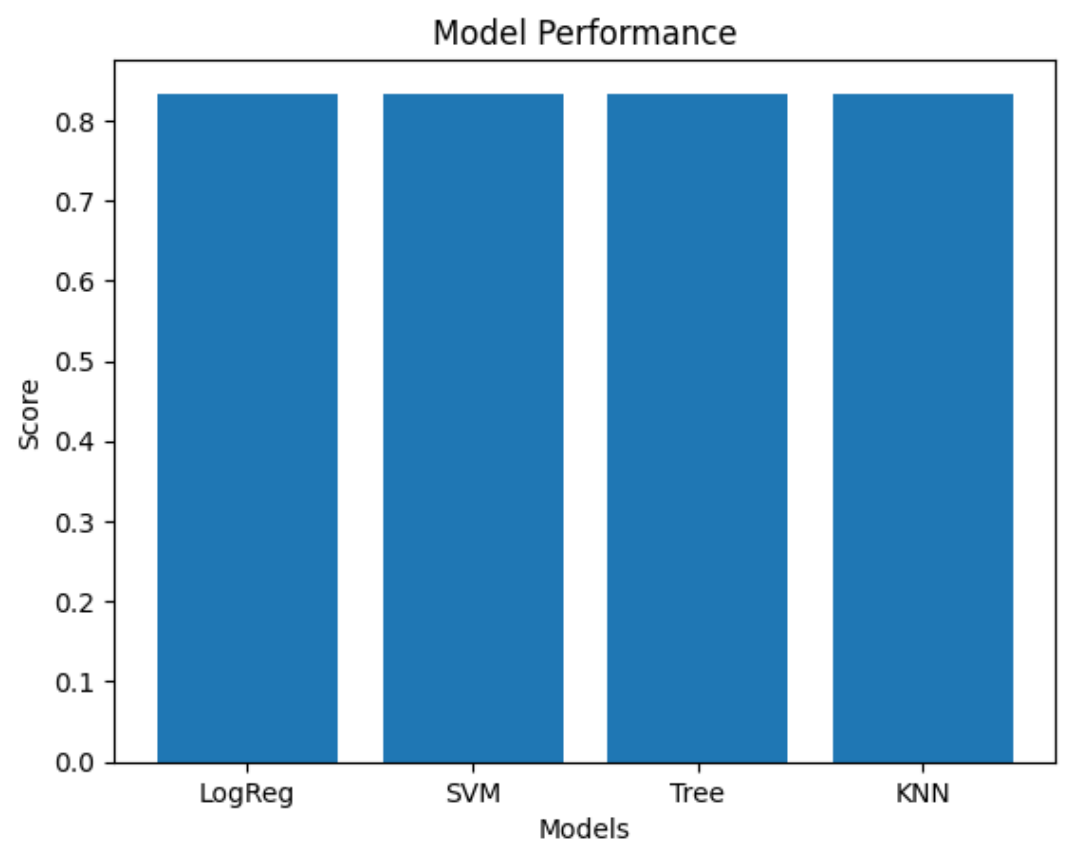




Section 5

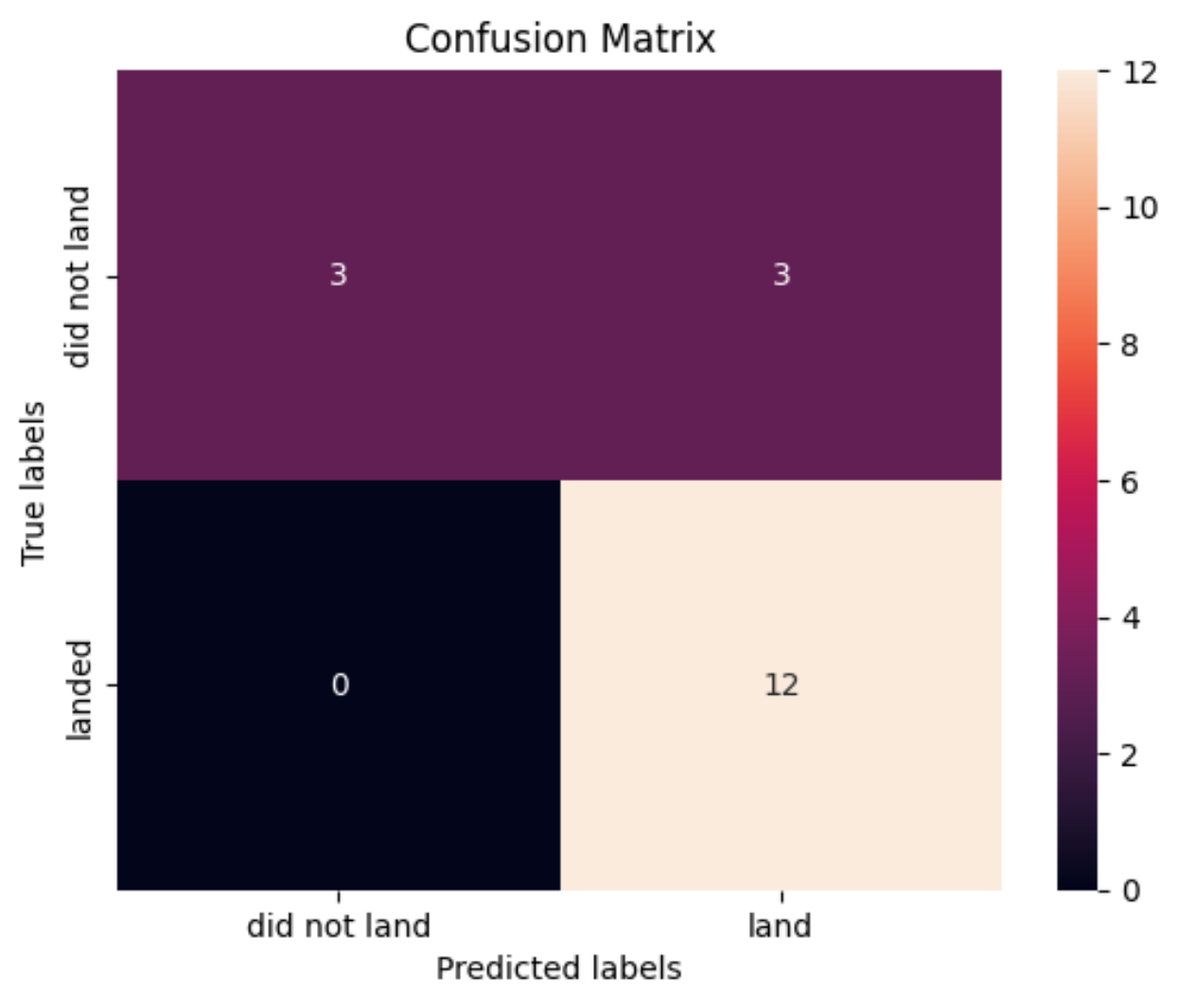
Predictive Analysis (Classification)

Classification Accuracy



- All four models had the same model efficiency: 0.833.
- The data set was small, so there was not much opportunity for divergent performance among models.

Confusion Matrix



- All four models had identical confusion matrices.

Conclusions

- Mean launch success rate was 2/3.
- There is a learning curve—success rate increased with time.
- Using actual time would be more informative than "flight number."
- Site CCAFS SLC 40 had a large number of launch failures, but it also had a large number of early flights.
- The effect of payload on launch success was ambiguous. One analysis pointed to heavy flights being more successful while another concluded that light flights were more successful.
- There is not enough data to conclude what orbit type is most successful.
- There is not enough data to choose a machine learning model. Results were identical.
- Visualization choices matter. Pie charts are rarely effective. Scatter plots are not effective in showing distributions.

Appendix

- I'd love to make more effective graphs like stacked bar charts and histograms, but I am out of time.
- Pie charts are not effective. Please stop using them.
- Scatter plots are not great for inferring distribution. Use histograms when you can.
- All code and output can be found in my Github Directory:

https://github.com/ANetTow/testrepo/tree/c32e04b89155b31ebee921cb14253c02f4a18d4d/Capstone

Thank you!